MARCH 18-22, 2024
SAN FRANCISCO, CA

# CSharpify Your Game Engine

*A GUIDE TO EMBEDDING C#*

## 👋 Andreia “shana” Gaita

spoiledcat.com

Hacker of Games, Ports, Tools, Runtimes, Libraries, Engines, Language bindings

shana@mastodon.gamedev.place

## C#

## The Glossary

Can’t have a conversation without words

## In the beginning, there was…

## In the beginning, there was…

Java?

## Java!

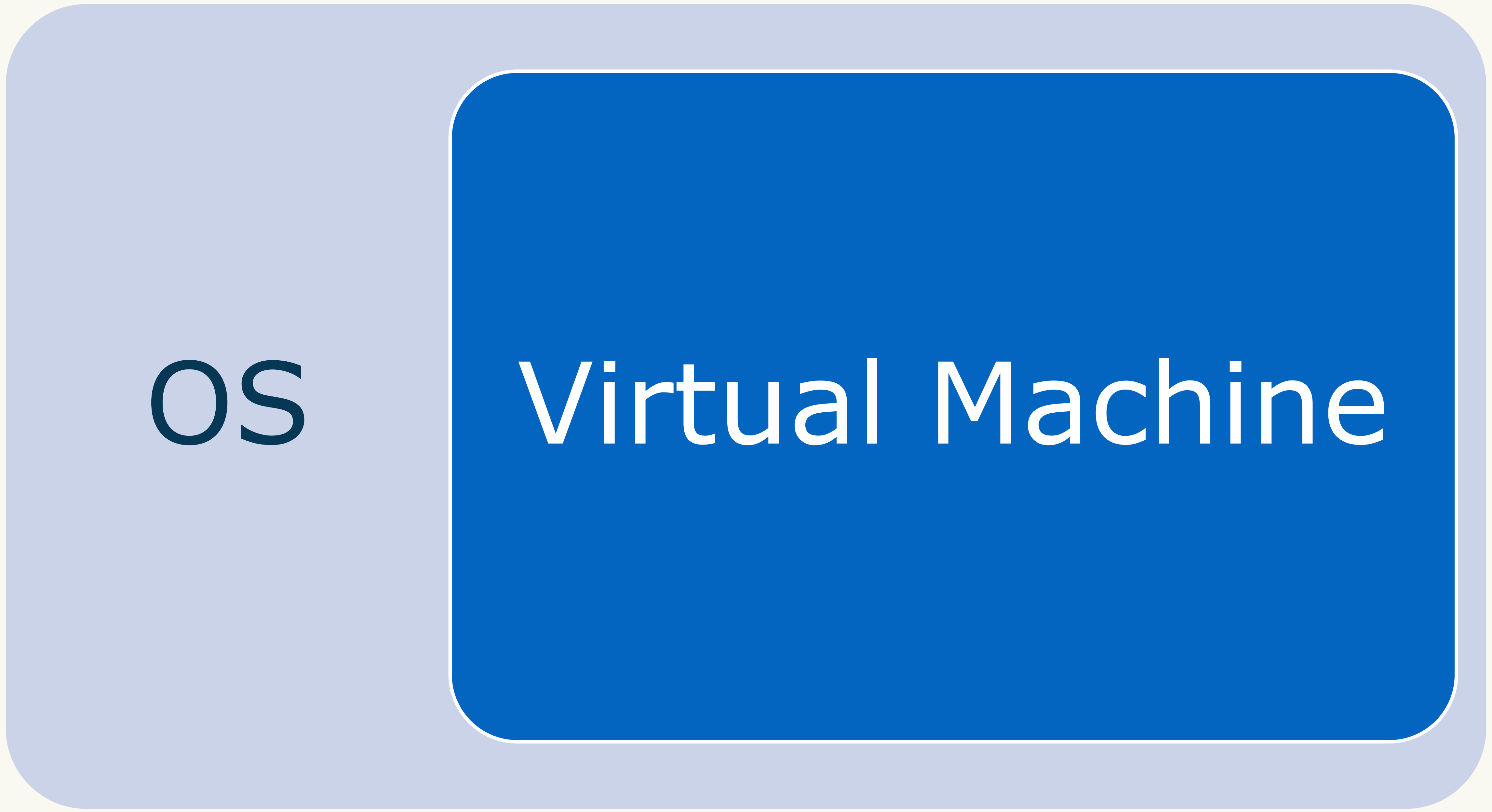
Write Once
Run Anywhere(tm)

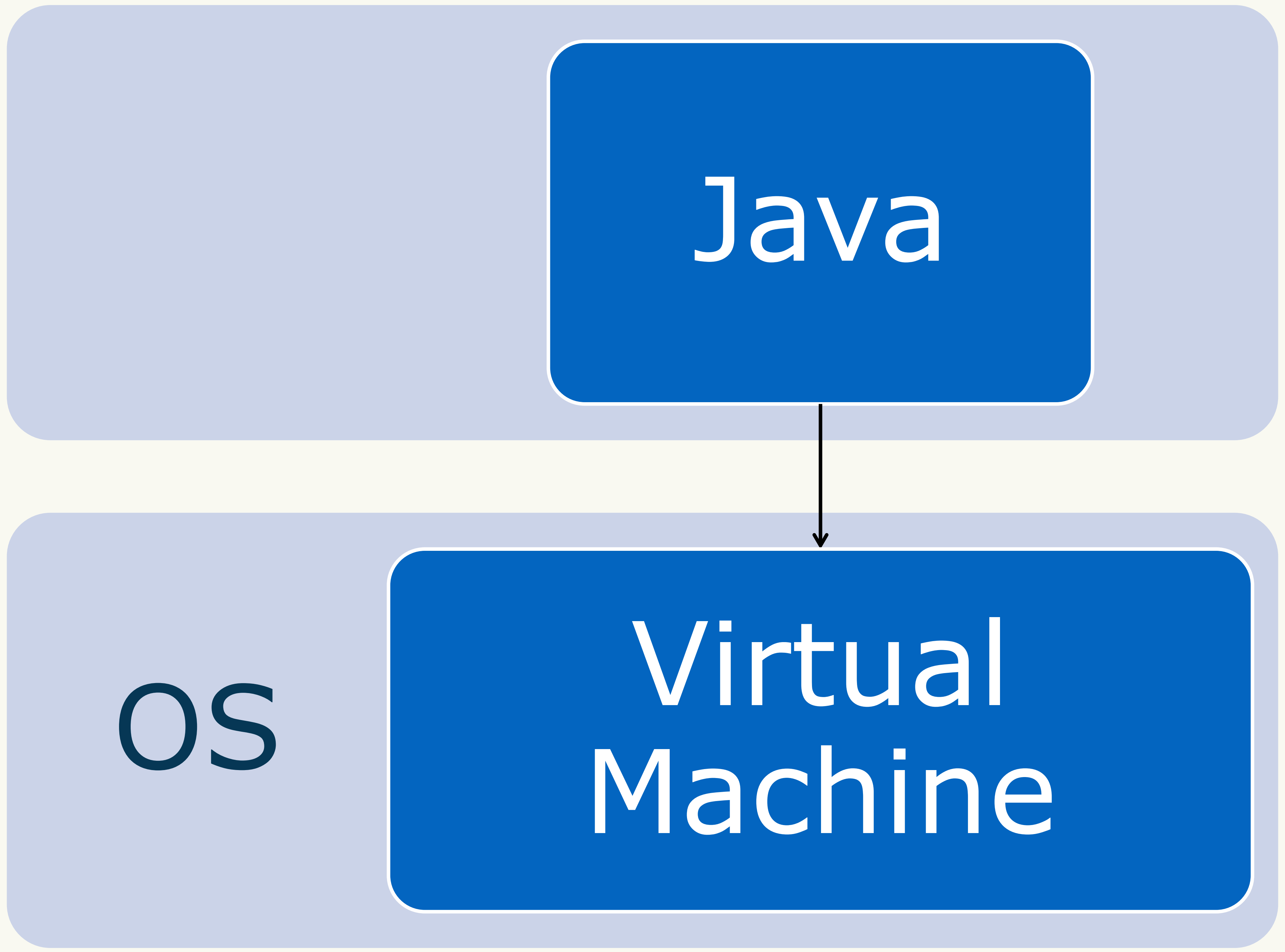
## Java

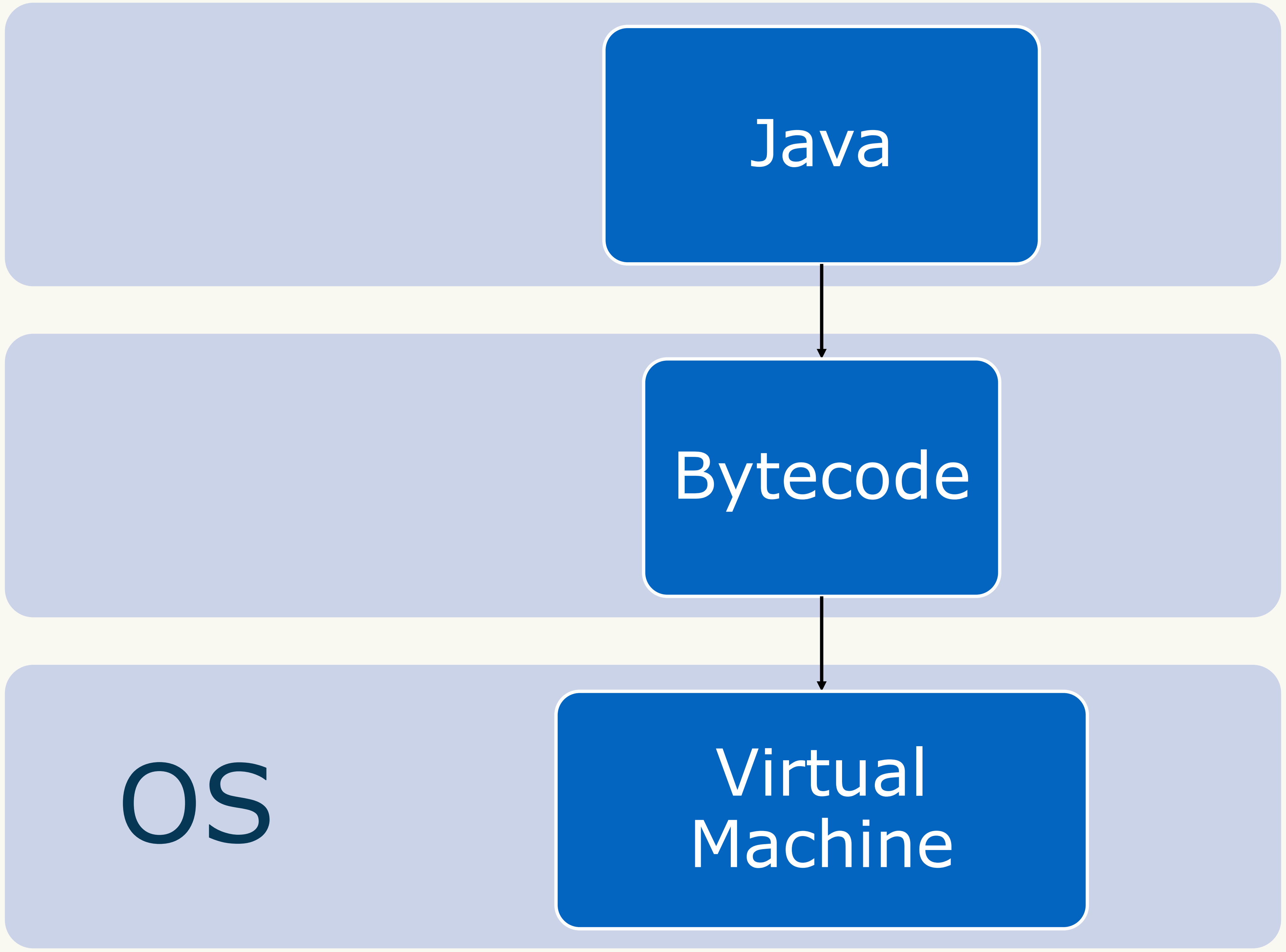
Write Once
Run Anywhere(tm)

Java

Virtual Machine


OS
Virtual Machine


Java
OS
Virtual Machine


Java
Bytecode
OS
Virtual Machine

## A Challenger Appears...

## A Challenger Appears...

Microsoft

## A Challenger Appears...

**Microsoft**

**Native Call Performance**

# Native Call Performance

- Win32 API
  - Must go fast!

## Native Call Performance

- Win32 API
  - Must go fast!

- Java’s native call performance was… poor
  - Every call assumed to be managed by the GC

Compiler

Native Calls

Bytecode

Garbage Collector

Virtual Machine

Pinning

Machine Code

## Native Call Performance

Visual J++

```java
/** @dll.import("USER32", entrypoint="GetSysColor") */
static native int GetSysColor(int nIndex);
```

*J++ J/Direct native call (1996-2004)*

```
[DllImport("user32.dll", CharSet=CharSet.Auto)]
static extern int GetSysColor(int nIndex);
```

*C# P/Invoke native call*

"Sun has responded to Microsoft's release of Internet Explorer (IE) 4.0, and its 2.0 release of the SDK for Java (SDKJ) with a lawsuit in U.S. District Court.
[...]
Microsoft made the choice [...] to ship products it claims are fully Java 1.1 compliant, but which failed to pass the Java 1.1 compatibility tests"

*What does Sun's lawsuit against Microsoft mean for Java developers?*
*JavaWorld, October 1 1997*

"Microsoft does not support the Java Native Interfaces (JNI) or the Remote Method Invocation (RMI), and it has altered the Core Java Class Libraries with about 50 methods and 50 fields that are not part of the public Java Application Programming Interfaces (APIs) published by Sun."

*What does Sun's lawsuit against Microsoft mean for Java developers?*
*JavaWorld, October 1 1997*

Compiler

Native Calls

Class Libraries

Bytecode

Garbage Collector

Virtual Machine

Pinning

Machine Code

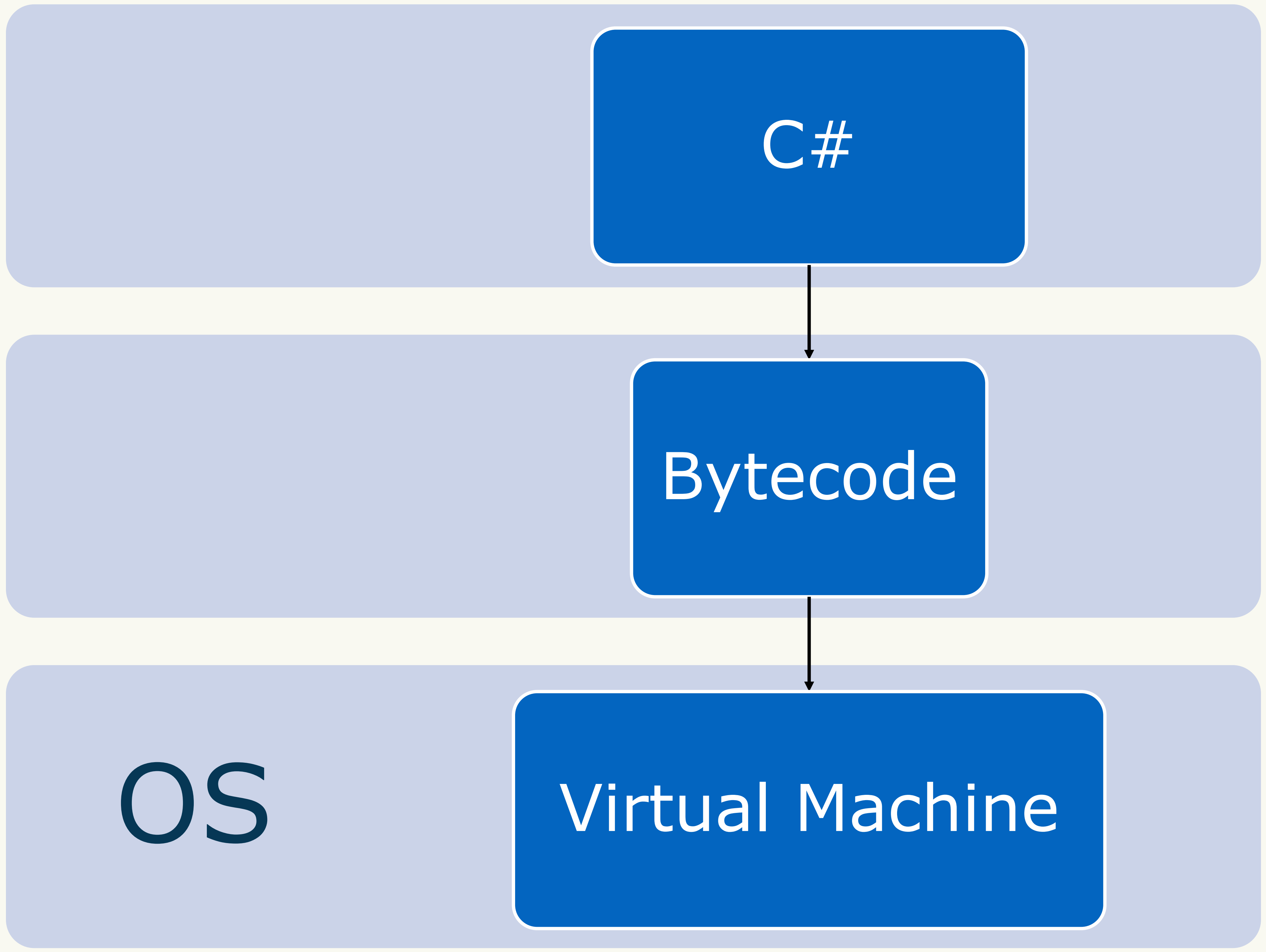
## Gotta Go Fast?

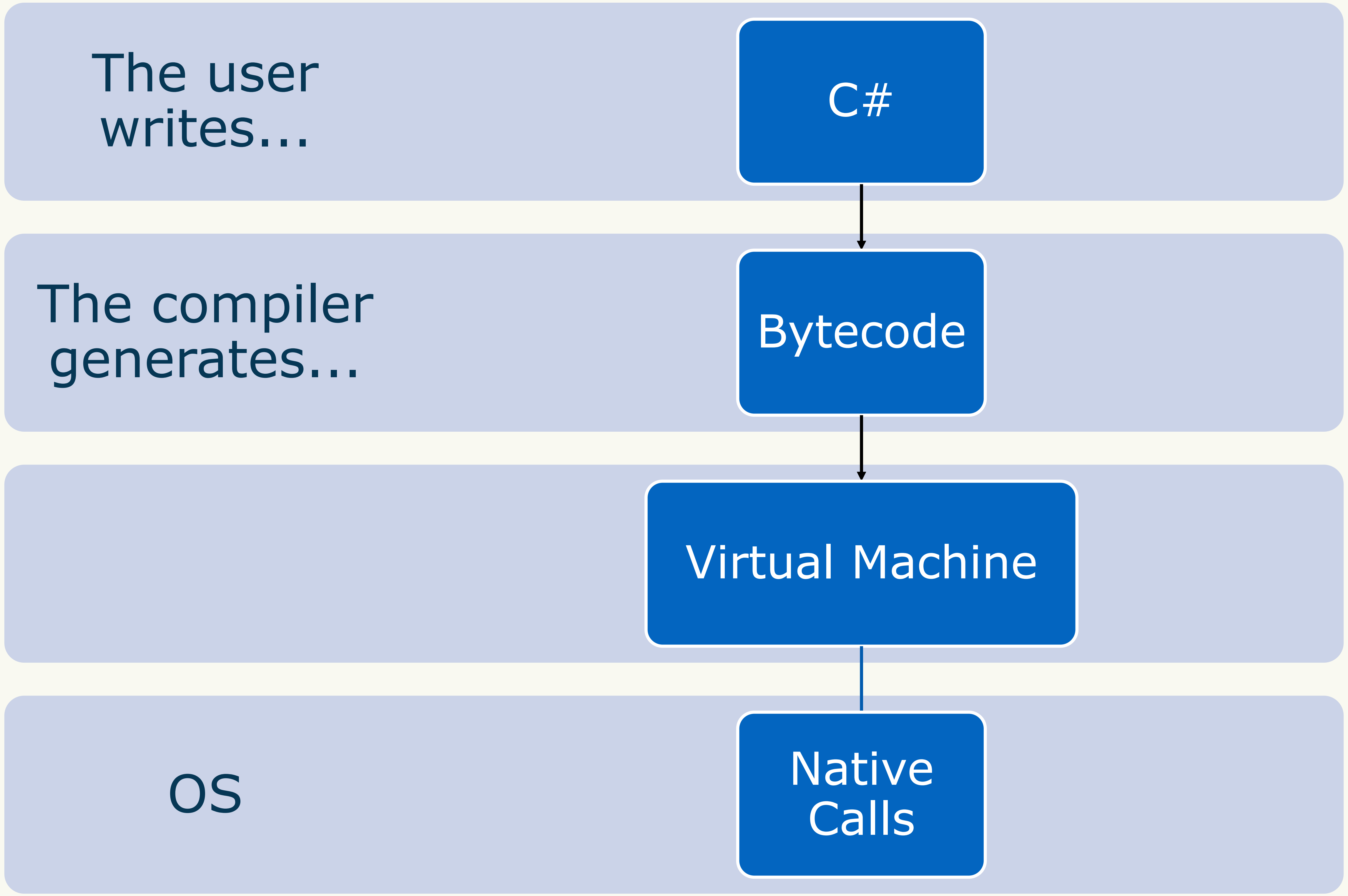
**Visual J++**

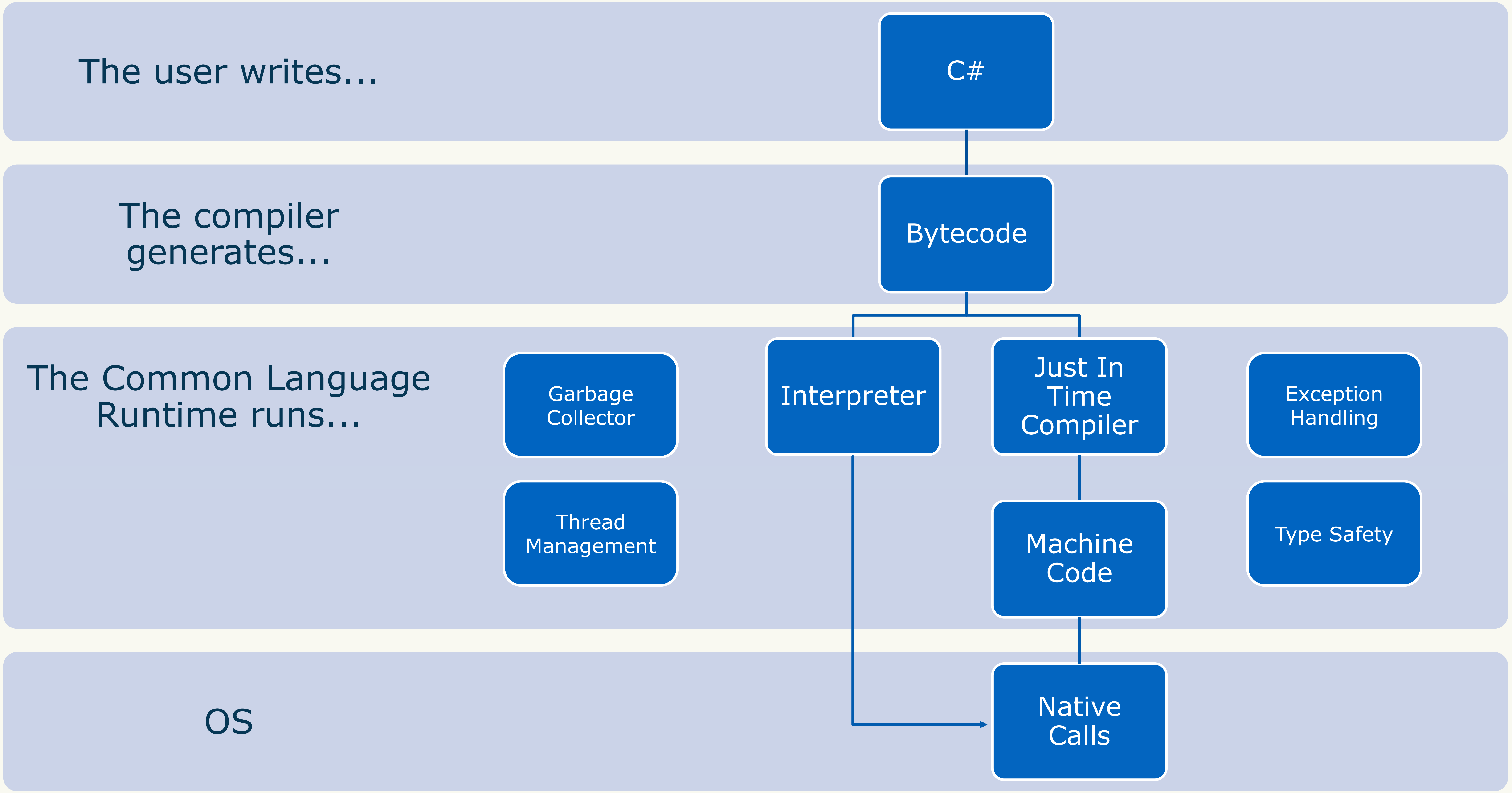
## Gotta Go Fast!

~~Visual J++~~

C#


C#
Bytecode
OS
Virtual Machine


The user writes...
C#
The compiler generates...
Bytecode
Virtual Machine
OS
Native Calls


The user writes...
C#
The compiler generates...
Bytecode
The Common Language Runtime runs...
Garbage Collector
Interpreter
Just In Time Compiler
Exception Handling
Thread Management
Machine Code
Type Safety
OS
Native Calls

Just In Time Compiler

C#

Compiler

Interpreter

Native Calls

Class Libraries

Bytecode

Garbage Collector

Virtual Machine

Common Language Runtime

Machine Code

Pinning

JIT

C#

Compiler

Interpreter

P/Invoke

Class Libraries

IL

GC

VM

CLR

Machine Code

Pinning

# .NET Framework

JIT

C#

Compiler

Interpreter

P/Invoke

IL

GC

Tools

CLR (the VM)

Class Libraries

## .NET

*OR HOW TO BE JUST SOOOO BAD AT NAMING THINGS*

## .NET, The Naming Saga

- .NET - The ecosystem, the brand
- .NET Framework - The language, tools, class libraries, CLR
  - First release in 2002
  - Last update in 2022
  - Versions 1.0 to 4.8.1

# Based on public standards

- Standard ECMA-334
  - C# Language Specification
  - 1st edition December 2001, 7th edition December 2023

- Standard ECMA-335
  - Common Language Infrastructure
  - 1st edition December 2001, 6th edition June 2012

- Patent Promises

# C# and .NET Framework

| C# Version | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 7.3 |
|---|---|---|---|---|---|---|---|---|
| .NET Framework Version | 1.0 | 2.0 and 3.0 | 3.5 | 4 | 4.5 | 4.6 | 4.7 | 4.8 |

## .NET, The Naming Saga

- We’re .NET Core now!
- First release in 2016
- Let’s reset the version!
    - ...But not the language version, don’t be silly!
- Versions 1.0 to 3.1

# C# and .NET Framework and .NET Core

| C# Version | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 7.1 | 7.3 | 8.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| .NET Framework Version | 1.0 | 2.0<br>3.0 | 3.5 | 4 | 4.5 | 4.6 | 4.7 | | 4.8 | |
| .NET Core Version | | | | | | 1.0 | | 2.0 | 2.1<br>2.2 | 3.x |

## .NET, The Naming Saga

- Let’s get ready for .NET Core 4!
- Wait what, people use embedded .NET Framework/Core version metadata in compiled DLLs for feature detection?
- Wait what, our tooling does that too??
  - Oh [censored]...
- It’s fine, let’s just skip version 4

## .NET, The Naming Saga

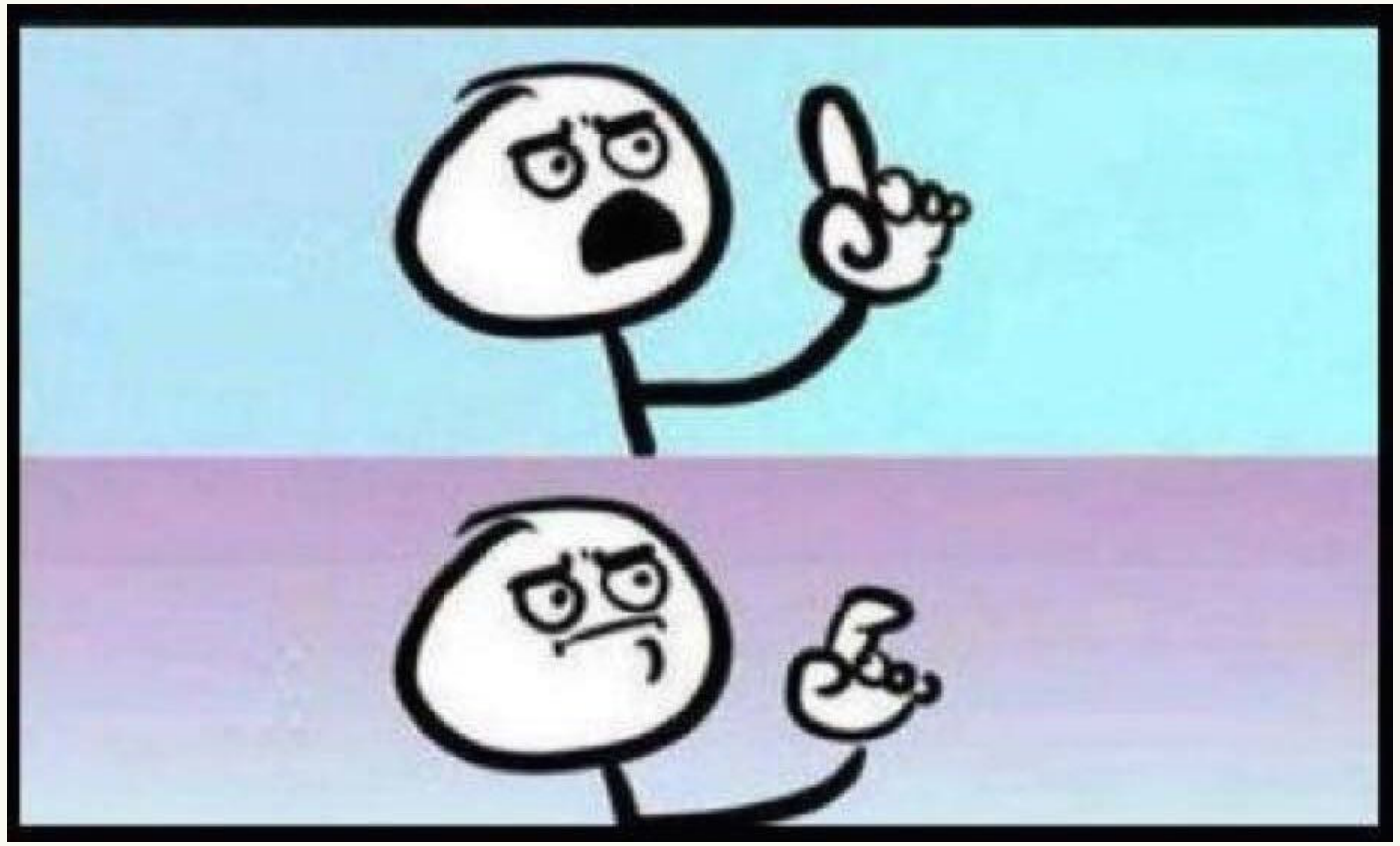
- Let's get ready for .NET Core 4!
- Wait what, people use embedded .NET Framework/Core version metadata in compiled DLLs for feature detection?
- Wait what, our tooling does that too??
  - Oh [censored]...
- It's fine, let's just skip version 4
- *high fives all around*

## .NET, The Naming Saga

- So wait, if we’re skipping version 4 of .NET Core… why not rebrand?
- Great idea! What should we call it?
- How about… .COM?
- Nah, .COM is already taken.
- How about… .NET?
- Hey, that’s a great idea!

## .NET, The Naming Saga

- So wait, if we’re skipping version 4 of .NET Core… why not rebrand?
- Great idea! What should we call it?
- How about… .COM?
- Nah, .COM is already taken.
- How about… .NET?
- Hey, that’s a great idea!
- *high fives all around*



## .NET, The Naming Saga

- What should we call our command line tool?
- How about...

*`dotnet`*

## .NET, The Naming Saga

- What should we call our command line tool?
- How about...

*dotnet*

- *high fives all around*

## .NET, The Naming Saga

- .NET is the ecosystem
- .NET is the tooling and CLR and class libraries
- *dotnet* is the command line tool

## .NET, The Naming Saga

- .NET is the ecosystem
- .NET is the tooling and CLR and class libraries
- *`dotnet`* is the command line tool

## C# and .NET Framework and .NET Core and .NET

| C# Version | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 | 7.0 | 7.1 | 7.3 | 8.0 | 9.0 | 10.0 | 11.0 | 12.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| .NET Framework Version | 1.0 | 2.0<br>3.0 | 3.5 | 4 | 4.5 | 4.6 | 4.7 | | 4.8 | | | | | |
| .NET Core Version | | | | | | 1.0 | | 2.0 | 2.1<br>2.2 | 3.x | | | | |
| .NET Version | | | | | | | | | | | 5.0 | 6.0 | 7.0 | 8.0 |

## .NET, But Less Insane

- .NET - the ecosystem
- .NET Core - the tooling and CLR and class libraries
- *`the dotnet tool`* – the command line tool

dotnet tool

JIT

C#

Compiler

Interpreter

P/Invoke

# .NET

IL

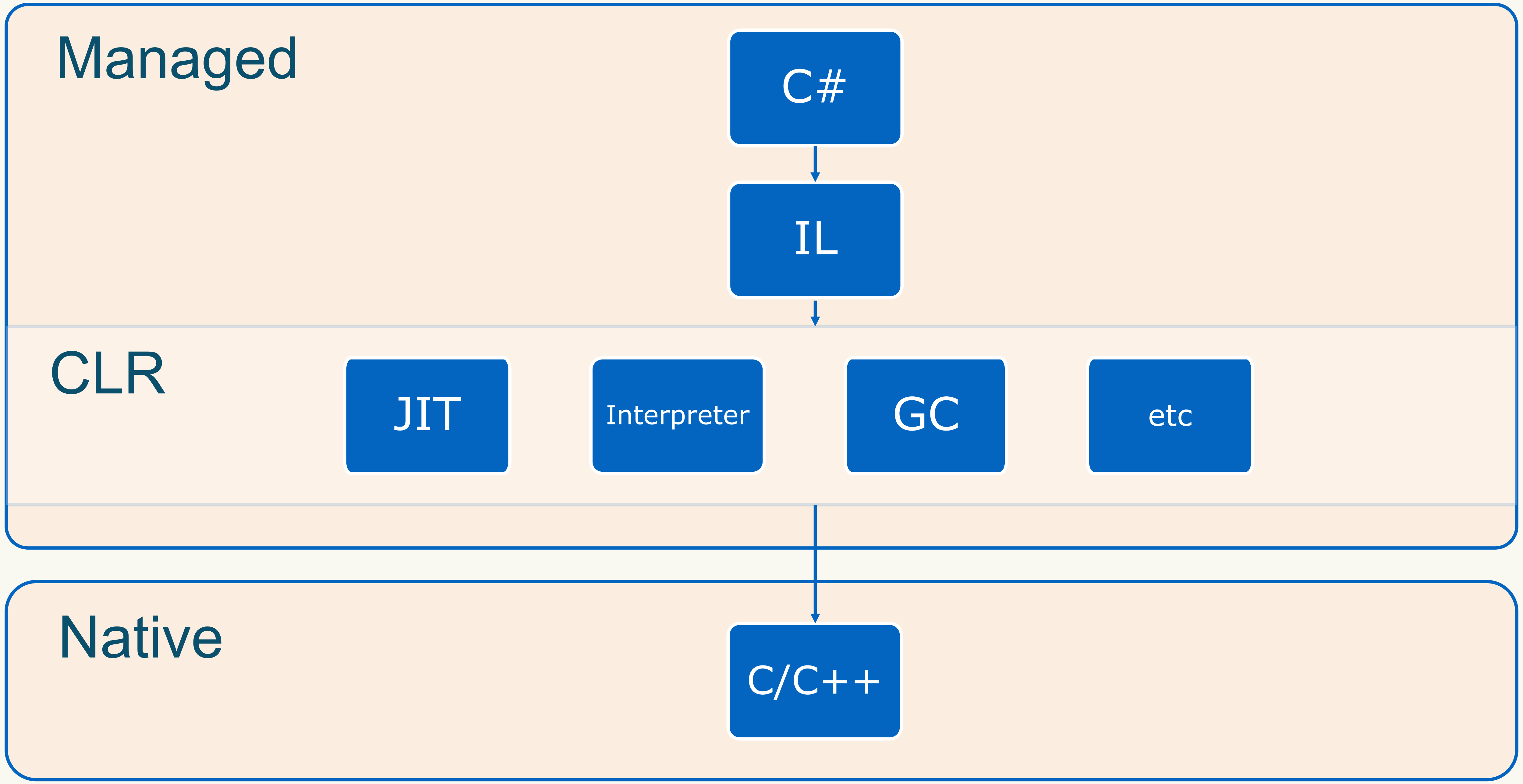
GC

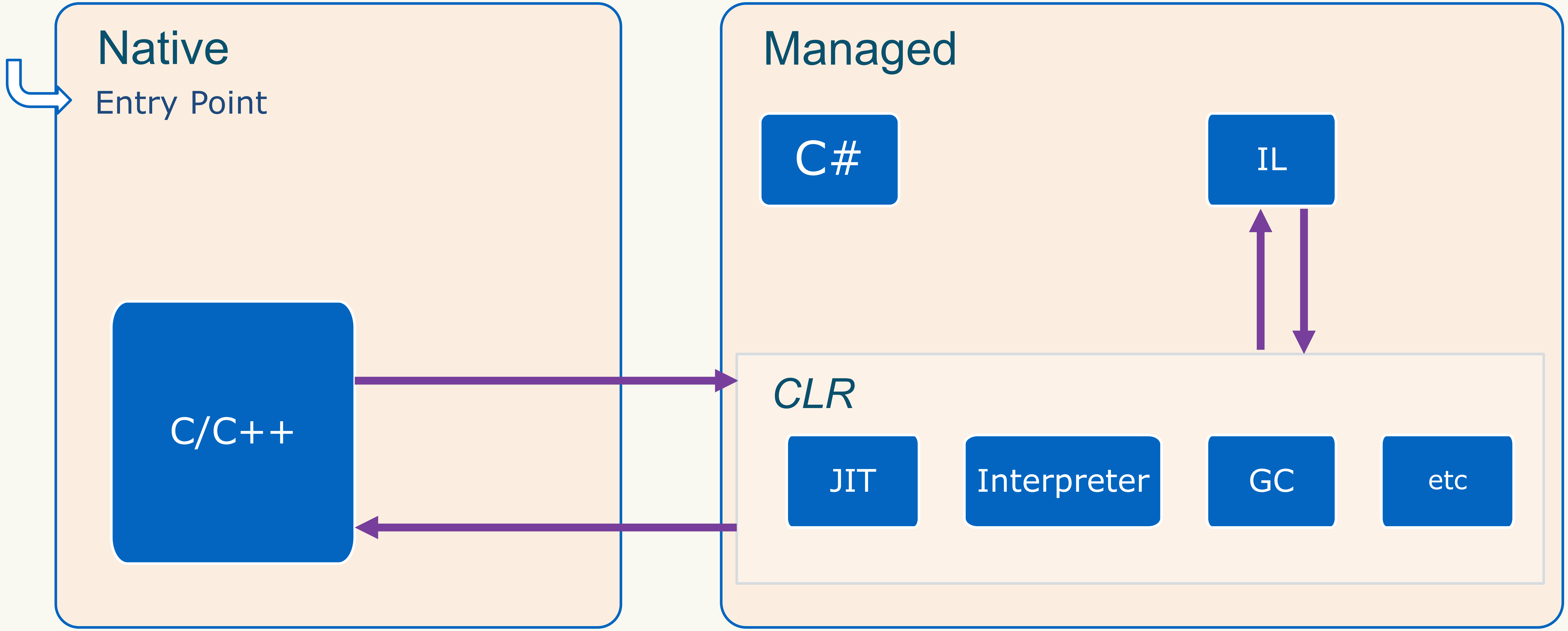
.NET Core

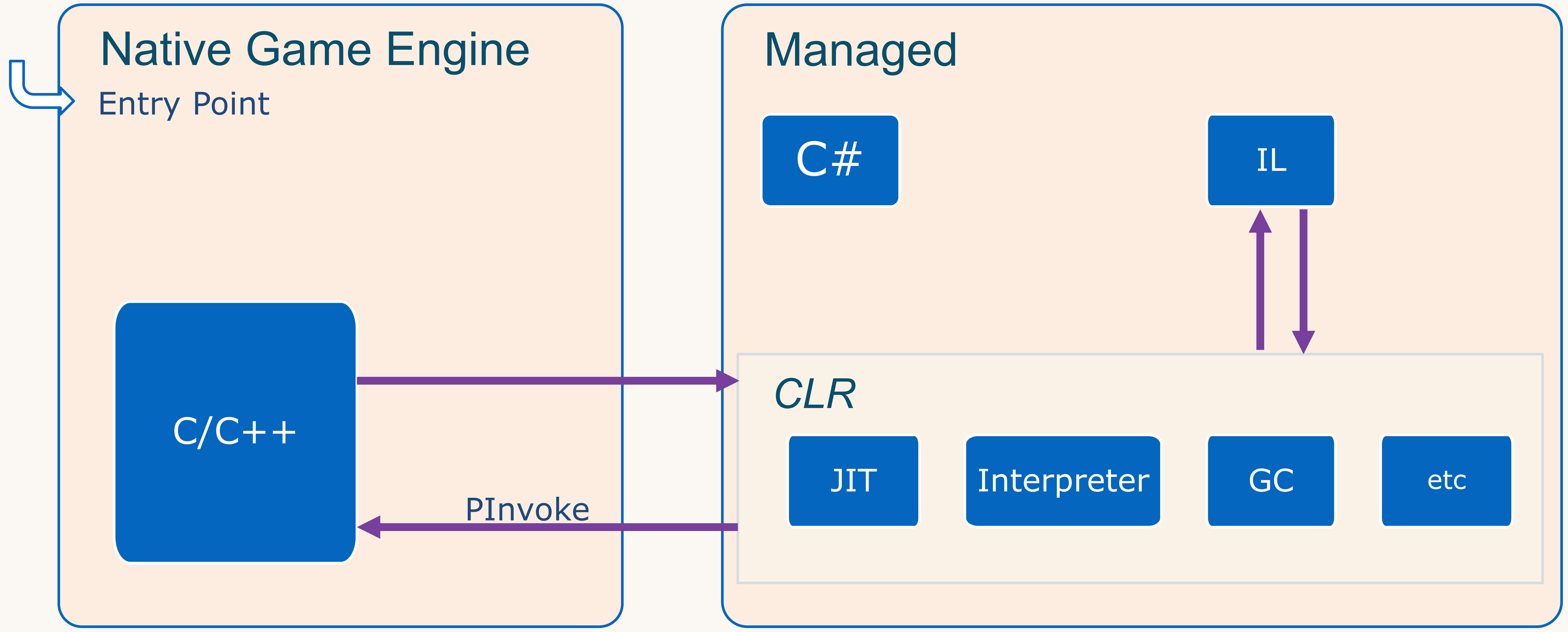
Tools

CLR
(the VM)

Class
Libraries

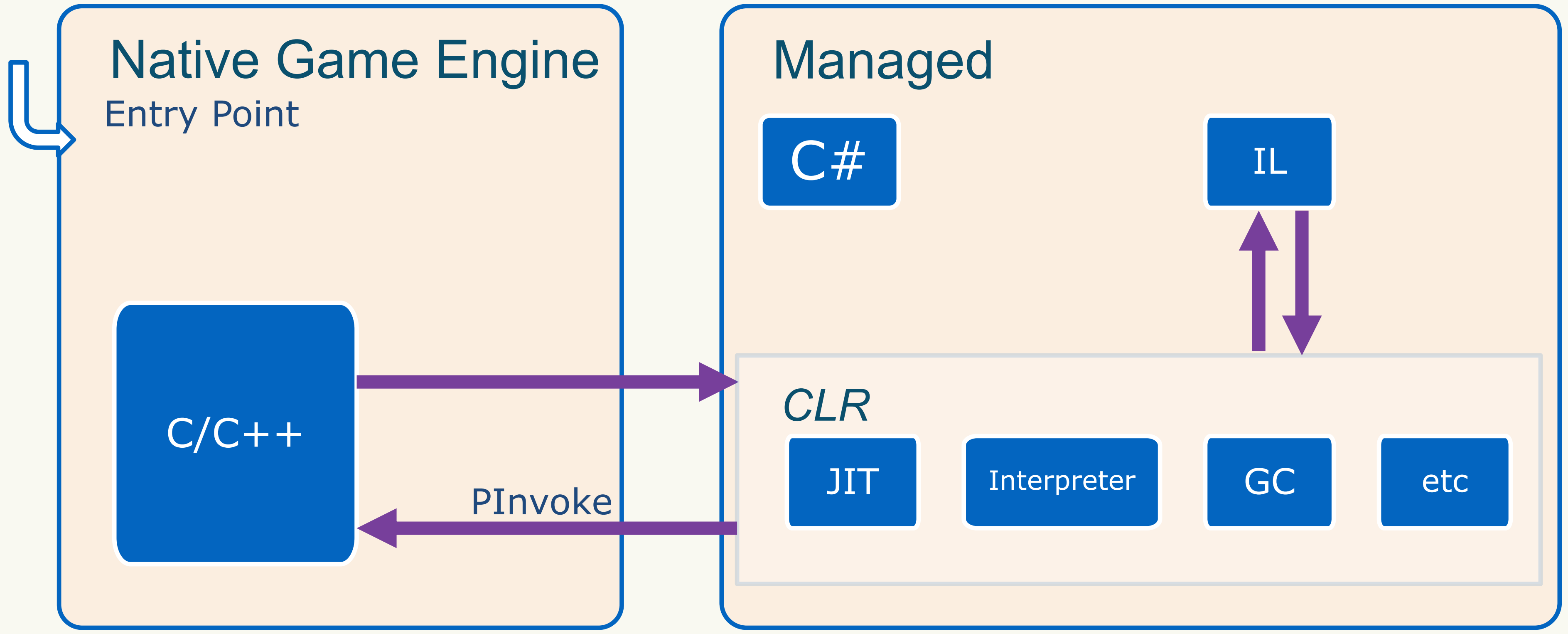

Managed
C#
IL
CLR
JIT
Interpreter
GC
etc
Native
C/C++


Native
Entry Point
C/C++
Managed
C#
IL
CLR
JIT
Interpreter
GC
etc


Native Game Engine
Entry Point
C/C++
PInvoke
Managed
C#
IL
CLR
JIT
Interpreter
GC
etc

# C# as an embedded language

- The Mono Project
- First released in 2004
- Clean-room implementation of the ECMA standards
- Cross-platform
- Open source
- *and* Ahead Of Time compiler - AOT



JIT

C#

Managed

AOT

Interpreter

Compiler

P/Invoke

# Mono

IL

GC

Native

Class Libraries

Tools

CLR (the VM)

## Mono

- Used everywhere
- Ported to everything
- Developed by Ximian, acquired by Novell
- MIT-licensed...
  - but the runtime was dual-licensed, either GPL or commercial

## Unity

- Popularizing C# in games
- Embedding Mono!
- With the runtime commercial licensed from Novell
- And all was well with the world!

## The Attachmate Problem

- Attachmate buys Novell
- Day of the merger, the entire Mono team is laid off
- Attachmate walks away with the whole Mono IP
- Support contracts?
- Mono commercial licensees like Unity?
- Mobile customers?

## The Attachmate Problem

- Attachmate buys Novell
- Day of the merger, the entire Mono team is laid off
- Attachmate walks away with the whole Mono IP
- Support contracts?
- Mono commercial licensees like Unity?
- Mobile customers?

- Oh well, sucks to be you I guess

## Xamarin is formed

- Perpetual license for all IP
- Stewardship of the Mono project
- Customers can relax, we got you!

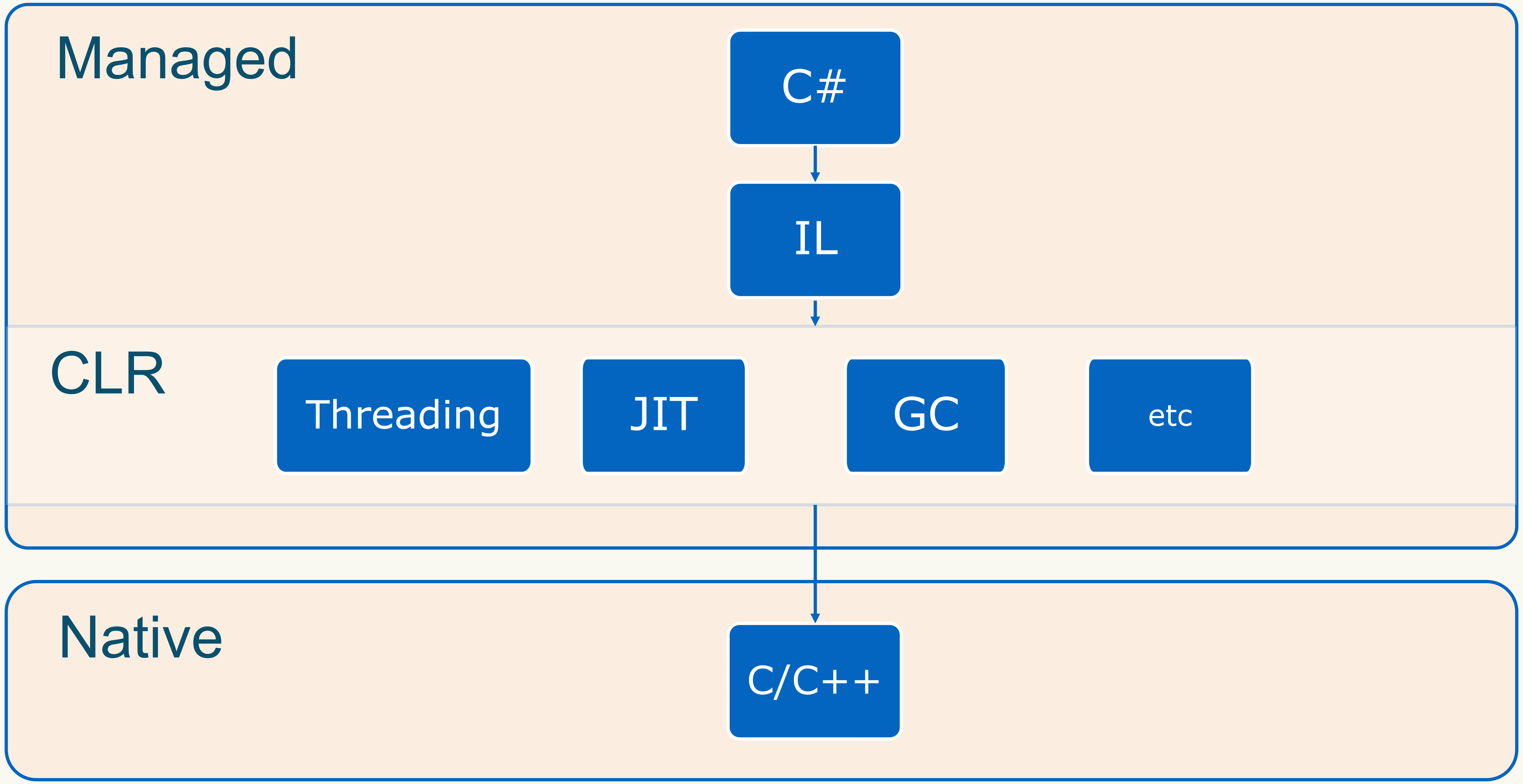
# Xamarin is formed

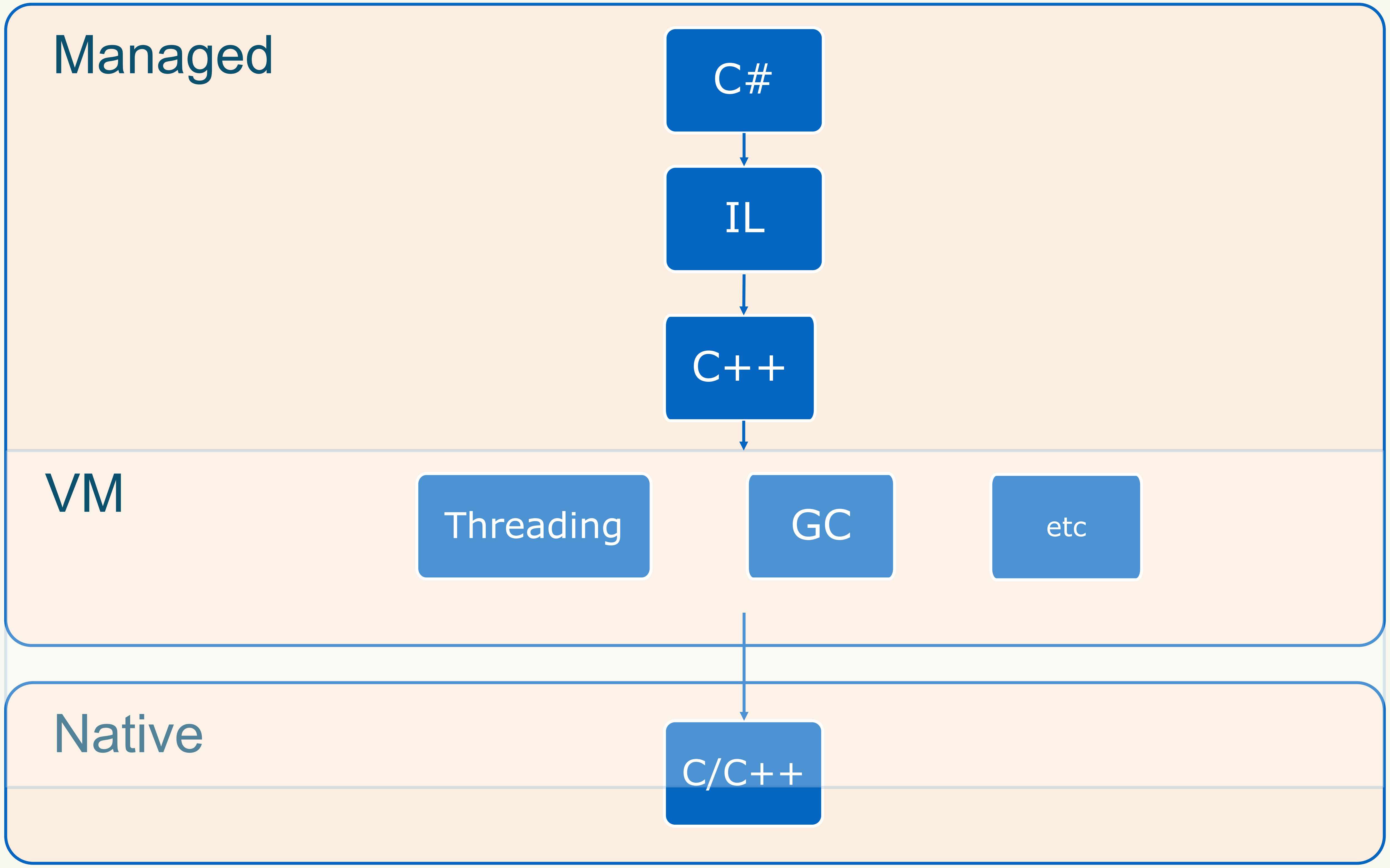
- Perpetual license for all IP
- Stewardship of the Mono project
- Customers can relax, we got you!

- ... except you Unity, you don’t get a license.

## Unity

- Mono runtime license up to 2011
- Apple doesn’t want GPL blobs in iOS
- ... and this is how we end up with IL2CPP


Managed
C#
IL
CLR
Threading
JIT
GC
etc
Native
C/C++



Managed
C#
IL
C++
VM
Threading
GC
etc
Native
C/C++

## Another .NET goes Open Source

- .NET Core
- Released in 2014
- CoreCLR, MIT Licensed
  - (and then renamed .NET, as we've already covered)

- Lives in https://github.com/dotnet/runtime

- ...not very useful for embedding

## Microsoft acquires Xamarin

- 2016
- Remaining Mono bits now fully MIT-relicensed
- Both .NET Core and Mono are moved to the .NET Foundation

## .NET Core and Mono, side by side

- Mono team contributes to both
- Mono included in the dotnet/runtime source
- .NET Core slowly gains proper cross-platform and embedding capabilities
- Mono slowly incorporates .NET Core improvements
- Unified tooling – it’s all *dotnet*

dotnet tool

AOT

JIT

C#

Managed

Interpreter

Compiler

P/Invoke

# .NET

Mono .NET Core

IL

Native

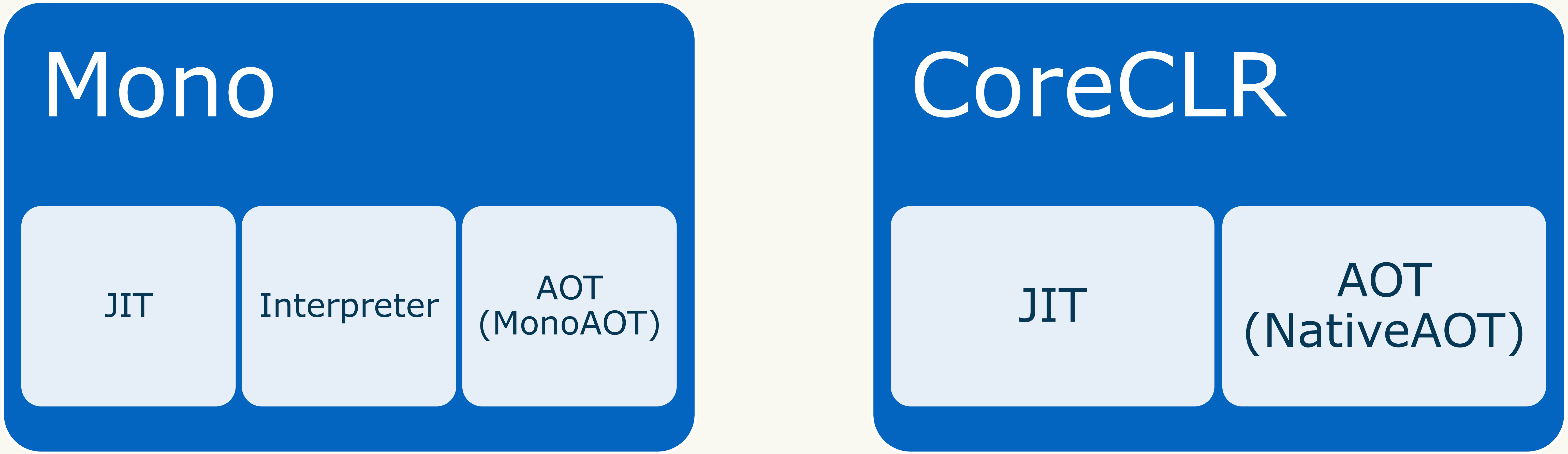
Class Libraries

GC

Tools

CLR (the VM)

## Embedding modes



AOT?

CoreCLR?

# But which one do I use?!?

JIT?

Interpreter?

Mono?

AOT?

CoreCLR?

Interpreter?

## All of them!

JIT?

Mono?

# All of them!

- Depends on the platform and available tooling
- win/mac/linux – All are available
- iOS – Interpreter or AOT
- Android – All are available – but Interpreter or AOT
- web – AOT, JIterpreter…
- Consoles – AOT or Interpreter (ping me for details)

## C# and C/C++ together

*LET'S GET TECHNICAL*

## Embedding steps

- Initialize the runtime
- Call C# methods from C/C++
- Call C methods from C#
- Pass data around as arguments and return values

## Setup

- 2 runtimes and 3 modes, but
  - Interpreter only on Mono
  - AOT has its own setup

- So, we have three broad setup types
  - Mono (JIT/Interpreter)
  - CoreCLR (JIT)
  - AOT

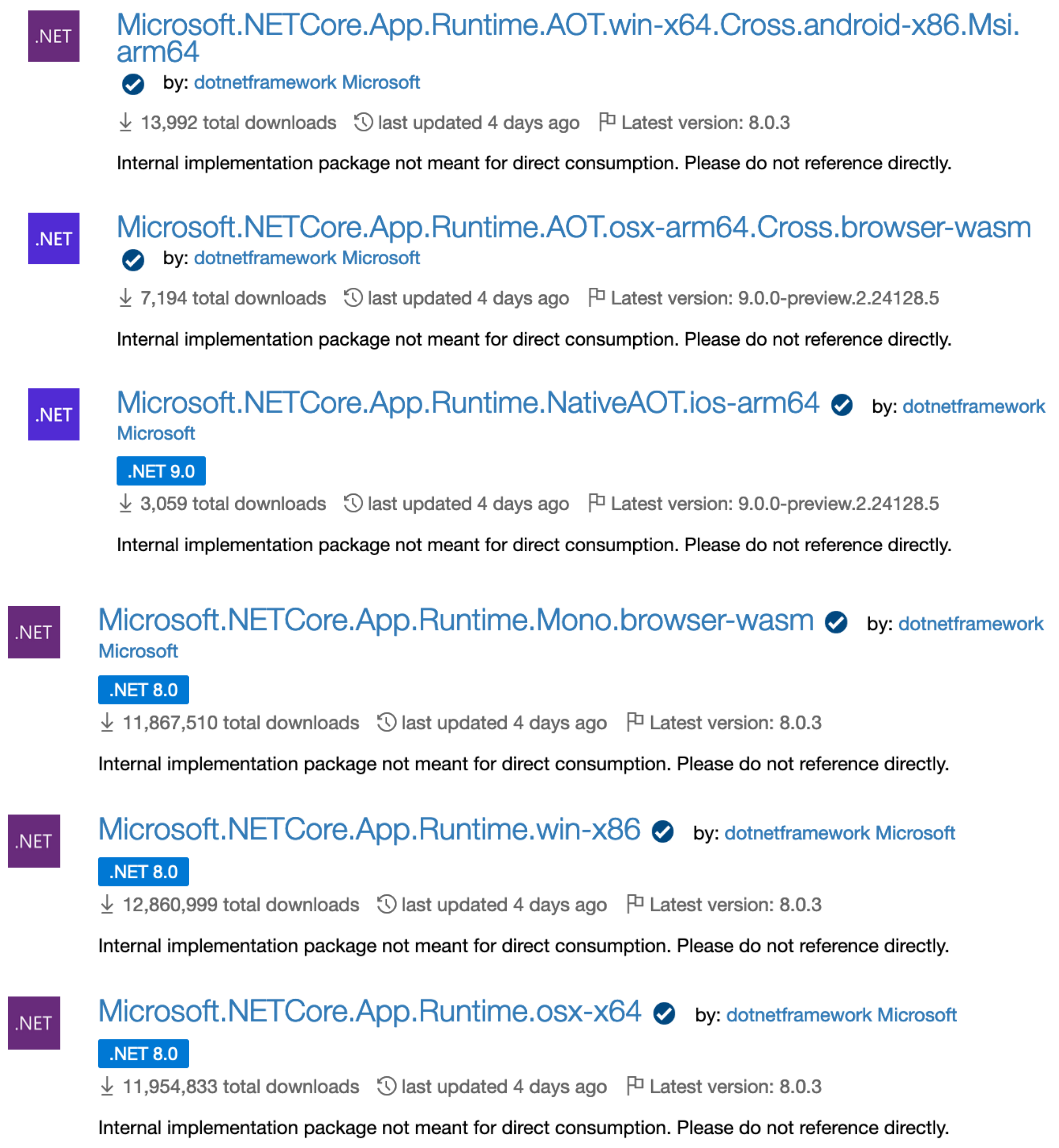
## Setup

- Runtimes expose C APIs
  - Mono – extensive C API
  - CoreCLR – the bare minimum C API

- Mono is a separate project, but a copy is in dotnet/runtime
  - Slowly adding additional APIs matching CoreCLR

## Setup

- Official packages for CoreCLR/Mono/AOT per platform all come from dotnet/runtime


Microsoft.NETCore.App.Runtime.AOT.win-x64.Cross.android-x86.Msi.arm64
by: dotnetframework Microsoft
13,992 total downloads last updated 4 days ago Latest version: 8.0.3
Internal implementation package not meant for direct consumption. Please do not reference directly.

Microsoft.NETCore.App.Runtime.AOT.osx-arm64.Cross.browser-wasm
by: dotnetframework Microsoft
7,194 total downloads last updated 4 days ago Latest version: 9.0.0-preview.2.24128.5
Internal implementation package not meant for direct consumption. Please do not reference directly.

Microsoft.NETCore.App.Runtime.NativeAOT.ios-arm64 by: dotnetframework Microsoft
.NET 9.0
3,059 total downloads last updated 4 days ago Latest version: 9.0.0-preview.2.24128.5
Internal implementation package not meant for direct consumption. Please do not reference directly.

Microsoft.NETCore.App.Runtime.Mono.browser-wasm by: dotnetframework Microsoft
.NET 8.0
11,867,510 total downloads last updated 4 days ago Latest version: 8.0.3
Internal implementation package not meant for direct consumption. Please do not reference directly.

Microsoft.NETCore.App.Runtime.win-x86 by: dotnetframework Microsoft
.NET 8.0
12,860,999 total downloads last updated 4 days ago Latest version: 8.0.3
Internal implementation package not meant for direct consumption. Please do not reference directly.

Microsoft.NETCore.App.Runtime.osx-x64 by: dotnetframework Microsoft
.NET 8.0
11,954,833 total downloads last updated 4 days ago Latest version: 8.0.3
Internal implementation package not meant for direct consumption. Please do not reference directly.

## CoreCLR Initialization

- It’s convoluted and complicated and annoying
- locate the hostfxr library
- call it to find the runtime
- pass a bunch of random strings
- or call...

```
coreclr_initialize
```

- but not on windows for some reason?

## Mono Initialization

- More fine-grained, but still somewhat envolved

```
monovm_initialize_preparsed
mono_install_assembly_preload_hook
mono_jit_init
mono_assembly_open
```

- ...and this is where we talk about *csharpify*

## github.com/spoiledcat/csharpify

- Example embedding C# in a “game engine”
  - Using Dear ImGUI + SDL2 + Vulkan

- Produces a header+source+cmake library that can be dropped into a project

## From C/C++ to C#

```
coreclr_create_delegate(coreclr_handle, coreclr_domainId,
"assembly", "MyType", "MyMethod", &delegate);
```

*CoreCLR delegate creation*

dotnet tool

AOT

JIT

C#

Managed

Interpreter

CLR

Compiler

P/Invoke

.NET

IL

Mono .NET Core

Native

delegate

VM

Class Libraries

GC

runtime

```
coreclr_create_delegate(nullptr, 0,
"assembly", "MyType", "MyMethod", &delegate);
```

*Mono delegate creation*

*C#*

```
[UnmanagedCallersOnly()]
static bool IsOk() { return false; }
```

*C/C++*

```
static bool (* IsOk_fnptr)(void);

IsOk_fnptr = (bool(*)(void)) coreclr_create_delegate(...);
```

dotnet tool

UnmanagedCallersOnly

AOT

Interpreter

Managed

JIT

CLR

C#

Compiler

blittable

P/Invoke

.NET

IL

Mono

.NET Core

Native

Class Libraries

GC

VM

delegate

runtime

trimming

## From C# to C/C++

```
[DllImport("MyLibrary")]
static extern bool IsOk();
```

*C#*

```
bool IsOk() { return false; }
```

*C/C++*

## [DllImport(“MyLibrary”)]

- Platform Invocation
  - P/Invoke for short

- Information about what to call and where to find it

- C functions only
  - because in C++, per-compiler name mangling is a thing

dotnet tool

UnmanagedCallersOnly

AOT

Managed

Interpreter

JIT

CLR

C#

Compiler

P/Invoke

.NET

blittable

IL

Mono

.NET Core

Native

DllImport

VM

Class Libraries

GC

delegate

runtime

extern

trimming

```
[DllImport(“MyLibrary”)]
static extern bool IsOk();

 ... or ...

[LibraryImport("MyLibrary")]
internal static partial bool IsOk();
```

*C# P/Invoke, since .NET 7*

dotnet tool

UnmanagedCallersOnly

AOT

Managed

Interpreter

JIT

CLR

C#

partial

Compiler

blittable

P/Invoke

.NET

IL

Mono

.NET Core

Native

DllImport

VM

LibraryImport

Class Libraries

GC

delegate

runtime

extern

trimming

## PInvoke, what happens?

1. The runtime allocates a chunk of unmanaged memory.
2. The managed class data is copied into the unmanaged memory.*
3. The unmanaged function is invoked, passing it the unmanaged memory information instead of the managed memory information.*
4. The unmanaged memory is copied back into managed memory.**

https://www.mono-project.com/docs/advanced/pinvoke/#memory-boundaries

# PInvoke, what happens?

2. The managed class data is copied into the unmanaged memory.*

3. The unmanaged function is invoked, passing it the unmanaged memory information instead of the managed memory information.*

* If it's a struct, it's on the stack, contains only blittable types, and is passed by reference, these steps are skipped.

https://www.mono-project.com/docs/advanced/pinvoke/#memory-boundaries

## PInvoke, what happens?

1. The runtime allocates a chunk of unmanaged memory.
2. The managed class data is copied into the unmanaged memory.*
3. The unmanaged function is invoked, passing it the unmanaged memory information instead of the managed memory information.*
4. The unmanaged memory is copied back into managed memory.**

https://www.mono-project.com/docs/advanced/pinvoke/#memory-boundaries

## PInvoke, what happens?

4. The unmanaged memory is copied back into managed memory.**

** Skipped for class (reference) types by default (can be modified by DllImport [Out] parameter

https://www.mono-project.com/docs/advanced/pinvoke/#memory-boundaries

## Marshalling Data

| Managed | Native |
|---|---|
| byte | uint8_t |
| short | int16_t |
| int | int32_t |
| long | int64_t |

https://www.mono-project.com/docs/advanced/pinvoke/#marshaling

## Value Types

- An instance of data
- Not tracked by the GC
- Passed around and returned by value (by default)
  - This means the contents of the thing are copied
  - If you pass a value type into a method, and change it, the change happens to the copy, not to the original

## Reference Types

- A pointer to an instance of data
- Tracked by the GC
- Passed around and returned by reference
  - This means there’s only one copy of the contents, and things that point to that content are passed around
  - If you pass a reference type into a method, and change the contents of it, that change is seen everywhere

## C# Struct

- A value type
- Allocated on the (stack or register or non-GC heap*)
- LayoutKind.Sequential by default
  - The layout of the fields of the struct matches the order in which they're declared
- If it contains non-blittable types (any field with a reference type, for eg), that triggers a copy during marshalling.

## C# Class

- A reference type
- Allocated on the GC heap – the GC managed memory pool
- LayoutKind.Auto
    - The order of the fields of the class is unknown – the runtime can rearrange it to optimize for access or space or whatever
- A pointer to a managed class passed to a native function is only valid until that function returns (i.e. don’t store it for later use)

## Memory and Reference Types

- Great for long lived objects
- Avoid fragmenting memory by allocating up front and reusing objects
  - Keep pressure low on the GC, so it doesn’t have to constantly track new objects
- References are tiny, negligible to copy around
- Access the same memory from anywhere

## Memory and Value Types

- Great for short-lived objects
- Great for marshalling
- Copied by value, so be careful with struct sizes

## Marshalling Data

| Managed | Native |
|---|---|
| byte | uint8_t |
| short | int16_t |
| int | int32_t |
| long | int64_t |
| bool | int32_t |

MARCH 18-22, 2024
SAN FRANCISCO, CA

```
[DllImport("MyLibrary")]
static extern bool IsOk();

void CheckIfOk() {
  if (!IsOk()) {
    Fail(); ← does this run?
  }
}
```

*C#*

```
bool IsOk() {
  return false;
}
```

*C/C++*

#GDC2024

```
[DllImport("MyLibrary")]
static extern bool IsOk();

void CheckIfOk() {
  if (!IsOk()) {
    Fail(); ← maybe, maybe not!
  }
}
```

*C#*

```
bool IsOk() {
  return false;
}
```

*C/C++*

```
Native execution
1111 1111 0000 0000 1100 1111 1001 1110  ← initial value of return location
1111 1111 0000 0000 1100 1111 0000 0000  ← "return false" sets 8 bits to zero

C#
bool = 1111 1111 0000 0000 1100 1111 0000 0000

if the return value is not all zeros, then it's true.
```

```
Native execution
1111 1111 0000 0000 1100 1111 1001 1110  ← initial value of return location
1111 1111 0000 0000 1100 1111 0000 0000  ← “return false” sets 8 bits to zero

C#
bool = 1111 1111 0000 0000 1100 1111 0000 0000

if the return value is not all zeros, then it’s true.
```

```
Native execution
1111 1111 0000 0000 1100 1111 1001 1110  ← initial value of return location
1111 1111 0000 0000 1100 1111 0000 0000  ← “return false” sets 8 bits to zero

C#
bool = 1111 1111 0000 0000 1100 1111 0000 0000

if the return value is not all zeros, then it’s true.
```

## Marshalling Data

- Know how managed types are converted to native types and vice-versa
  - Search online for "Type Marshalling"[1]

- bool is a trap, avoid it

[1] https://learn.microsoft.com/en-us/dotnet/standard/native-interop/type-marshalling

## DllImport – Function name

C#

```
[DllImport(“MyLibrary”, EntryPoint=“IsOk_Fixed”]
static extern bool IsOk();
```

C++

```
int32_t IsOk_Fixed() { return (int32_t)false; }
```

## Putting it all together

- Create managed-native and native-managed C# signatures
- Source generation with DNNE
- Implement extern “C” native functions
- Load runtime ahead of time or on first C# call

dotnet tool

UnmanagedCallersOnly

AOT

Interpreter

JIT

partial

C#

Managed

Bytecode

Blittable

Marshalling

reference type

Compiler

CLR

P/Invoke

.NET

IL

Mono

.NET Core

Native

DllImport

Pinning

GC

delegate

VM

LibraryImport

Class Libraries

trimming

extern

runtime

value type

Questions?

github.com/spoiledcat/csharpify

# THANK YOU!

github.com/shana

shana@mastodon.gamedev.place

@sh4na

https://spoiledcat.com

shana@spoiledcat.com

Andreia Gaita

## References

https://github.com/ocornut/imgui

https://github.com/shana/DNNE

https://aka.ms/dotnet-discord

# References

https://ericlippert.com/2010/10/11/debunking-another-myth-about-value-types/

https://jonskeet.uk/csharp/memory.html

https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/performance/ref-tutorial

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing?source=recommendations

https://github.com/mono/CppSharp/issues/1687 (Use LibraryImportAttribute instead of DllImportAttribute)

https://github.com/dotnet/runtime/issues/7267 (Support for Mono's DllImport(@"__Internal")?

https://www.mono-project.com/docs/advanced/pinvoke/

https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke

https://github.com/dotnet/samples/blob/main/core/interop/source-generation/custom-marshalling/src/custommarshalling/ErrorData.cs

https://learn.microsoft.com/en-us/dotnet/standard/native-interop/tutorial-custom-marshaller?source=recommendations

# References

https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.libraryimportattribute?view=net-8.0